**ASSIGNMENT 8**

ARITRA KUMAR LAHIRI

ASU ID: 1207275819

CSE 530 EMBEDDED OPERATING SYSTEM INTERNALS: PROJECT 2c:

**Adding storage capacity to the driver:**

1) There are 3 different ways by which we can add storage capacity to the driver such as kmalloc, vmalloc and k\_mem\_cache\_alloc.

2)The **kmalloc()** functions similar to that of a user's space malloc() routine, with the exception of an addition of a flags parameter. The **kmalloc()** function is a simple interface for obtaining kernel memory in byte-sized chunks.

The function is declared in <linux/slab.h>:

void \* kmalloc(size\_t size, int flags) // example of the function declaration

The function returns a pointer to a region of a memory that is at least **size** bytes in length. The region of memory allocated is physically contiguous. On error that is failure to allocate specified amount of memory due to unavailability the function returns null.

However kmalloc() is generally used for allocating small sized memory. But it has certain advantages than vmalloc() such as it does not go to sleep while allocating memory and the memory allocation is pretty fast.

3) The **vmalloc()** function works in a similar fashion to kmalloc(), except it allocates memory that is only virtually contiguous and not necessarily physically contiguous. The vmalloc() function only ensures that the pages are contiguous within the virtual address space. It does this by allocating potentially non contiguous chunks of physical memory and "fixing up" the page tables to map the memory into a contiguous chunk of the logical address space. The general syntax of the vmalloc() function declaration is:

void \* vmalloc(unsigned long size)

The function returns a pointer to at least size bytes of virtually contiguous memory. But generally we do not use vmalloc() because of its certain disadvantages such as :

a) vmalloc() allocation can go to sleep while allocating memory.

b) vmalloc() allocation is pretty slow due to TLB effects.

c) vmalloc() can only be used during init of modules, not during "normal operation", since it is really slow in allocation and free as well.

d) vmalloc() cannot be directly used for direct memory access. Hence for all these reasons vmalloc() is generally avoided unless we need a very big allocation which is very rare.

4) The kmem\_cache\_alloc() function is an implementation part of a look aside cache in kernel which facilitates the allocation of high volume objects that are most frequently used by the kernel. A device driver often ends up allocating same sized objects for a multiple number of times. Thus kernel have a special set of memory pools which facilitates allocation of its high volume objects through look aside cache. The cache manager is often called as slab allocator and the kmem\_cache() function is declared under <linux/slab.h>.

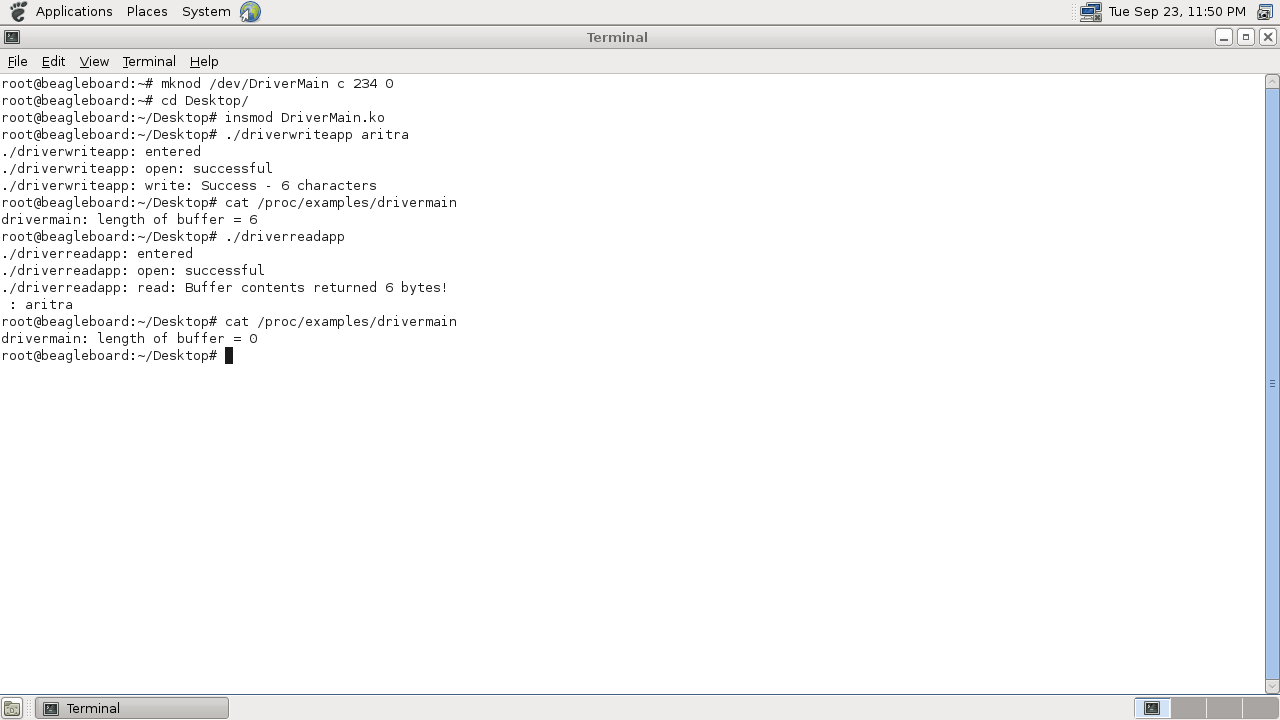
Once a cache of objects is created, you can allocate objects from it by calling kmem\_cache\_alloc:

void \*kmem\_cache\_alloc(kmem\_cache\_t \*cache, int flags); //function declaration

Here, the function keeps some copies of some pre-defined size objects pre-allocated. For example if we have struct operator that we know we will be requiring very frequently, so instead of allocating it from the main memory (kmalloc) when we need it, we already keep multiple copies of it allocated & when we want it, it returns the address of the block already allocated saving a lot of time. Similarly, when we free it, we don't give it back, it actually is not freed, it goes back to the allocated pool so that if some process again asks for it, we can return this address of the already allocated struct.

So comparing all of the above functionalities and the performance consequences I am using the kmem\_cache\_alloc() function to alter the storage capacity of a driver by adding characters from the write operation in a linked list.

The following are the screenshots for the use of the drivers operations that I have implemented using kmem\_cache\_alloc():



The following is the screenshot showing the write content of the buffer and the content of the buffer that gets removed after the read operation. We implemented that by using the /proc file system that returns length of the content of the buffer.